Productivity Engineering in the UNIX† Environment

Speed and Data Structures in Computer Algebra Systems

Technical Report

S. L. Graham
Principal Investigator

(415) 642-2059

"The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government."

Contract No. N00039-84-C-0089

August 7, 1984 - August 6, 1987

Arpa Order No. 4871

†UNIX is a trademark of AT&T Bell Laboratories

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION The Regents of the University of California 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION SPAWAR
6c. ADDRESS (*City, State, and ZIP Code*) Berkeley, California 94720	7b. ADDRESS (*City, State, and ZIP Code*) Space and Naval Warfare Systems Command Washington, DC 20363-5100
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)

* SPEED AND DATA STRUCTURES IN COMPUTER ALGEBRA SYSTEMS

12. PERSONAL AUTHOR(S)

* Richard J. Fateman, Carl G. Ponder

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
technical	FROM _____ TO _____	* August 31, 1987	* 8

16 SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Enclosed in paper.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (*Include Area Code*) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Speed and Data Structures in Computer Algebra Systems

Richard J. Fateman
Carl G. Ponder
Computer Science Division
University of California
Berkeley, CA. 94720

August 31, 1987

Abstract

Comparing the speed of computation in algebra systems is a perennial occupation of system designers, algorithm implementors, and, more recently, marketing personnel. At least some people have observed that for many problems, the choice of a system makes much less difference than the approach used to represent the problem. The mapping from mathematics to a data representation and the choice of algorithms can make significant, and separate, contributions to efficiency. Systems which have the flexibility to provide several data structures and algorithms can provide an advantage in this respect. Macsyma [1] is probably the system with the largest selection, currently. On the other hand, Macsyma has not taken advantage of recent advances such as the extensive use of hash-coding incorporated in the University of Waterloo's Maple [2] system. For the one somewhat artificial benchmark we discuss in this paper, it appears that the Maple system does considerably better than any representation in Macsyma by precisely this mechanism.

1 Introduction

This paper was inspired by a problem posed in sales literature as an appropriate benchmark for algebra systems. Consider the sequence of expressions defined by $f_0 = x$ and $f_n = \log(f_{n-1})$ for $n > 0$. Now compute various derivatives of the f_i's.

There are two major components to the cost of computing this result: the application of the chain rule, and the simplification or reordering of the terms in the result. These costs can be separated out and individually reduced. A good design is one in which no redundant applications of the chain rule are needed,

and in which terms are naturally produced in an ordered arrangement. In such a case the computation cost is about as low as possible. Of course, considering the fact that the answer is fairly bulky, there is a certain irreducible cost.

2 Some programs

We begin by considering how to arrange this calculation in Macsyma three different ways. The first is to use the most obvious representation. The second is to examine the structure of the problem in more detail and choose a different representation: declare a set of functions and their derivatives. The third way is to combine the canonical rational function package of Macsyma with the previous approaches. In Macsyma, the briefest way we have come across to request the 6th derivative of f_5:

```
f[0]:x $
f[n]:=log(f[n-1]) $
diff(f[5],x,6)$
```

Using the general "default" representation implied by these statements, the answer is clumsy, and so are the intermediate expressions. Macsyma does not recognize most of the common subexpressions within the object being manipulated and therefore repeatedly differentiates the various nested logarithms. In its determination to simplify the result (and intermediate expressions), Macsyma is faced with ordering 5th iterated logs, 4th iterated logs, etc, and must repeatedly traverse the subexpressions to find the depth of nesting. On a VAX8600 computer running UC Berkeley's "vaxima" implemented in Franz Lisp Opus 42, this takes 33.7 seconds (plus an additional 11.3 seconds in "garbage-collection" for reclaiming storage)[1]. The times for computing the sixth derivatives of f_1 through f_8 appear as curve "naive" in the first figure.

A second approach is to explicitly save the representation of the f functions and merely inform Macsyma of the gradients of each of them. This takes more effort, but is a step in the right direction, computationally:

```
(gradef(f1(x),1/x),
gradef(f2(x),1/f1(x)*diff(f1(x),x)),
gradef(f3(x),1/f2(x)*diff(f2(x),x)),
gradef(f4(x),1/f3(x)*diff(f3(x),x)),
gradef(f5(x),1/f4(x)*diff(f4(x),x)))$
diff(f5(x),x,6)$
```

[1] We re-ran these experiments on a a Sun Microsystems Sun-3/75, and the time taken on the Sun is twice as long as the VAX 8600.

Now one advantage of this representation is the answer appears more compactly in terms of the `f` functions. Another advantage is that the execution time is somewhat shorter. It is easier for us or for the computer program to compare `f5` and `f4`, which wear their "depths" on their faces, so to speak, and therefore the simplification time is reduced. But not by much. The differentiation time is not greatly reduced because each time the gradient of an `f`-function is needed, the gradient definition is applied, and therefore the derivative of `f4`, for example, is repeatedly computed. The time for this on our VAX 8600 is 26 seconds (plus 7.6 seconds in garbage collection). The time required by this method for other examples is shown by curve "gradef" in the first figure.

A major step in the speedup is one which would cut down this repetition in computing the derivatives. It turns out that the rational function package in Macsyma computes with non-rational "kernels" fairly faithfully, and in differentiating, computes the derivatives of kernels like `f4`, only once. The fact that there is a unique representation for the derivatives, speeds the computation considerably. Also contributing to the speed is the rational function manipulation package's canonical internal representation which leads to a trimmed-down algorithm for differentiation, and simplification. The answer is also more compact.

All that is required to use rational simplification is the conversion of some basic data item to `rat` form. Since the form is contagious, all the rest will follow. Alter the last line of the previous command to

```
diff(rat(f5(x)),x,6)$
```

The time has now been reduced dramatically to 3.03 seconds (plus 3.02 seconds in garbage collection). This time is shown as curve "rat" in the first figure.

Here is another simplification of the problem[2]. We have already reduced a sequence of transcendental functions to a sequence of rational functions in several variables. We can go one step further and reduce the rational functions to polynomials. That is, the denominators can all be unity if we represent `1/x` by `xi`, and similarly represent `1/f` functions by `fi`. We also have written out the gradient definitions, saving an additional 0.1 second on this run.

```
(gradef(xi(x),  -xi(x)^2),
  gradef(f1i(x),-f1i(x)^2*xi(x)),
  gradef(f2i(x),-f2i(x)^2*f1i(x)*xi(x)),
  gradef(f3i(x),-f3i(x)^2*f2i(x)*f1i(x)*xi(x)),
  gradef(f4i(x),-f4i(x)^2*f3i(x)*f2i(x)*f1i(x)*xi(x)),
  gradef(f5i(x),-f5i(x)^2*f4i(x)*f3i(x)*f2i(x)*f1i(x)*xi(x)),
  gradef(f5(x),  f4i(x)*f3i(x)*f2i(x)*f1i(x)*xi(x)))$
```

[2] *A useful rule of thumb in using algebraic systems is to try to convert the problem to a simpler domain, to improve performance.*

```
diff(rat(f5(x)),x,6)$
```

Now the time is down to only 1.65 seconds (plus 1.5 more in garbage collection), so we have reduced the effective CPU time by a factor of 20. These appear in the first figure as curve "naive + long gradef" in Macsyma's default general representation, and as curve "rat + long gradef" using rational functions.

This program has become a bit clumsy, and along the way we have discovered that by using the canonical rational expression package we've saved a fair amount of time. What happens if we abandon most of our cleverness, return to to our first program, and by using **rat** coerce the computations to being in rational form?

```
f[0]:x$
f[n]:=log(f[n-1])$
diff(rat(f[5]),x,6)$
```

This time is 3.5 seconds (plus 3.2 more in garbage collection). That is, we can get a factor of almost 10 merely by this minor alteration and no mathematical analysis. The time for this method is shown as curve "naive + rat" in the first figure.

The second figure compares the fastest and slowest methods of computing various derivatives of a fixed nested log. The fastest method is that given by the third program above. The slowest is the first program.

3 Comparison to Maple

A significant question is how much of the advantages of this speedup can be automatically conferred upon a calculation by an astute system. Clearly one heuristic in Macsyma is to try rational representations when possible. It is possible to construct problems in which this slows down the computation, but this occurs rarely. In the case of algebraic algorithms used in this example, two techniques are compared encoding expressions for uniqueness thereby avoiding redundant values plus the saving of derivative values. Both of these ideas are used in the Maple system. How does our best time of 1.7 seconds compare to Maple?

The simplest Maple program we came up with (including the printing of timings) looks like this:

```
f := proc (x,n) if (n=0) then x else ln(f(x,n-1)) fi; end:
st:= time():
answer:=diff(f(x,5),x,x,x,x,x,x):
time()-st;
```

Maple's time for this on a Sun-3 is an impressive 1.67 seconds. Thus on a VAX 8600 we would expect it to be about 0.8 seconds. Although these times tend to be difficult to reproduce exactly, it appears that for this problem, Maple is about 2 to 5 times faster than Macsyma when the Macsyma user "tunes" the code and as much as 40 times faster when Macsyma is used in the most natural and naive formulation.[3]

4 Additional Notes

We have been studying this particular contrived problem and other calculations for potential execution by multiple processors. Introducing hash-coding or some other technique for recognizing unique subexpressions has implications not only for serial computation, but for multi-processor speedups. That is, a problem which can be naturally divided into different components each stored in a hash-table can be operated on in a tabular rather than tree-like organization. We expect this to be a more natural organization for a multiprocessor system.

As an additional note, when we first took up this problem, we thought it would be interesting to try to formulate parallel processing approaches for this computation.

Of all the formulations presented here, the slowest appears likely to benefit most directly by parallel processing, where the massive redundant simplifications could be done in parallel. Had we not looked further to improve the serial algorithm, we could have written a paper on achieving substantial speedups by parallelism. It is unlikely that a significant speedup can be found for the better formulations unless we develop an algorithm for computing the 6th derivative which does not first compute the 5th (etc.).

5 Conclusion

Choice of algorithms and data structures represent critical issues for symbolic computing systems. Although one cannot always expect such dramatic differences as here, the default automatic choice for this problem is far better in Maple than in Macsyma; Macsyma can, however, be prompted to make an improved choice.

6 Acknowledgments

This work was supported in part by the Army Research Office, grant DAAG29-85-K-0070, through the Center for Pure and Applied Mathematics, University of California, Berkeley, and the Defense Advanced Research Projects Agency

[3]The naive formulation on Macsyma is not the slowest system we have tried: Mu-math, running on an IBM PC-XT is about 2000 times slower than Maple on a Sun-3.

(DoD) ARPA order #4871, monitored by Space & Naval Warfare Systems Command under contract N00039-84-C-0089, through the Electronics Research Laboratory, University of California, Berkeley.

References

[1] Fateman, R.J. (June 20-28, 1978). Macsyma's General Simplifier: Philosophy and Operation. In Lewis, V.E. (ed) *Proceedings of the 1978 Macsyma Users Conference*. Washington D.C., 563-582.

[2] Char, B.W. et al. (June 1984). *On the Design and Performance of the Maple System*. Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, CS-84-13.

Figure 1 -- Time in ms. to compute 6th derivative of nth nested log(x)

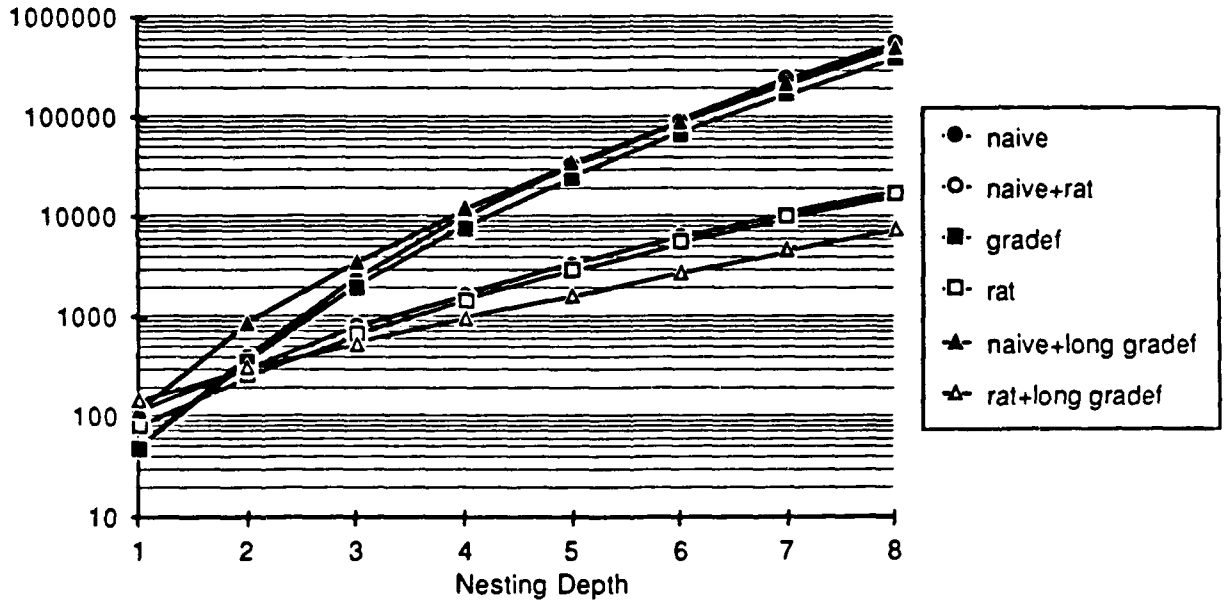

Figure 2 -- Time in ms. to compute nth derivative by fast & slow methods